educação, formação & tecnologias

Uma proposta para ajudar alunos com dificuldades na aprendizagem inicial de programação de computadores

ANABELA GOMES
Instituto Superior de Engenharia de Coimbra; Centro de Informática e Sistemas da Universidade de Coimbra,
anabela@isec.pt

JOANA HENRIQUES
Centro de Informática e Sistemas da Universidade de Coimbra,
joanahenriques33@hotmail.com

ANTÓNIO JOSÉ MENDES
Centro de Informática e Sistemas da Universidade de Coimbra,
toze@dei.uc.pt

Resumo: Os elevados níveis de insucesso em disciplinas onde são ensinados os conceitos mais básicos de programação, em qualquer grau e sistema de ensino, é um problema universal que tem sido alvo de variadas pesquisas, resultando também em diversificados sistemas, sem que contudo o panorama tenha melhorado significativamente. Na nossa óptica existe um conjunto de razões que estão na origem do problema, nomeadamente, métodos de ensino e aprendizagem desadequados, falta de vários tipos de competências por parte dos alunos, em particular no que respeita à resolução de problemas, a difícil natureza do tema e uma forte conotação negativa que lhe está associada. Porém, pensamos que as metodologias tradicionalmente utilizadas para aprender/ensinar estes assuntos não se revelam suficientes, pelo que é apresentada uma nova proposta, centrada na actividade de resolução de problemas, de acordo com o nível cognitivo do aluno e os seus estilos preferenciais de aprendizagem.

Palavras-chave: Ensino e Aprendizagem da Programação, Estilos de Aprendizagem, Psicologia da Programação.

1. INTRODUÇÃO

No seu nível mais elevado de proficiência a programação é muito mais do que a escrita de um conjunto de linhas de código numa dada linguagem, é uma arte e uma ciência. Arte porque existem muitas maneiras diferentes de codificar instruções, com alguma criatividade. É também uma ciência, porque é constituída por um conjunto de regras orientadoras, porque é necessário o uso de lógica e porque existem alguns métodos rigorosos de programação que asseguram a eficiência, economia e utilidade dos programas gerados.

A um nível mais básico, o ensino das linguagens de programação tem como propósito conseguir que os alunos desenvolvam as suas capacidades, adquirindo os conhecimentos básicos necessários para conceber programas capazes de resolver problemas reais simples. Porém, a este nível, existem

enormes taxas de insucesso nas disciplinas de programação onde estes aspectos básicos são ensinados. A experiência tem demonstrado que existe, em termos gerais, uma grande dificuldade em compreender e aplicar certos conceitos abstractos de programação, por parte de uma percentagem significativa dos alunos que frequentam disciplinas introdutórias nesta área. Uma das grandes dificuldades reside precisamente na compreensão e, em particular, na aplicação de noções básicas, como as estruturas de controlo, para a criação de algoritmos que resolvam problemas concretos.

Existem opiniões variadas e por vezes divergentes no que respeita às causas para tal insucesso, em função das quais têm surgido diferentes ferramentas com o propósito de minimizar essas dificuldades. Embora a avaliação da utilização dessas ferramentas frequentemente mostre impactos positivos no desempenho de alguns estudantes, as taxas de evasão e repetência em disciplinas de programação continuam elevadas.

Neste artigo são discutidas várias causas apresentadas na literatura, que contribuem para o insucesso do ensino/aprendizagem de programação, problemática que nos tem levado a reflectir sobre uma série de questões que também apresentaremos. Será igualmente apresentada uma nova proposta que pretende contribuir para minimizar este problema.

2. OS PROBLEMAS DE APRENDIZAGEM DE PROGRAMAÇÃO – NA LITERATURA

A literatura apresenta diversas justificações para a dificuldade inerente ao aprender a programar. Dijkstra (1989) argumenta que este tipo de aprendizagem é um processo lento e gradual. Almeida et al. (2002) referem que se observa, neste tipo de matérias, falta de interesse por parte dos alunos. Referem adicionalmente que esta desmotivação está associada a uma forte carga de conceitos abstractos que intervêm em todo o conhecimento envolvido na actividade de programação, onde as características próprias das linguagens e ambiente de programação, cada vez mais sofisticados e da máquina em si, tendem a dificultar a programação. Também Motil e Epstein (2000) afirmam que a maioria das linguagens de programação utilizadas nas disciplinas introdutórias apresenta uma sintaxe grande e complexa, mais adequada para ambientes de desenvolvimento industrial.

De acordo com Dijkstra (1989) e Perkins et al. (1988) a aprendizagem de programação requer um treino intensivo em resolução de problemas, envolvendo competências de diversas áreas para obter um pequeno retorno. Os autores afirmam ainda que, em simultâneo, se exige uma precisão e atenção a detalhes muito mais elevada do que a requerida pela maioria das outras disciplinas. Porém, autores como Byrne e Lyons (2001) referem que não há nada inerentemente difícil no assunto mas, simplesmente, existem alunos que não têm as aptidões necessárias para programar, nomeadamente de resolução de problemas e de matemática. Igual opinião é manifestada em Júnior e Rapkiewicz (2004, 2005).

Jenkins (2002) refere várias causas do insucesso generalizado em disciplinas de programação, como sejam o baixo nível de abstracção, a falta de competências de resolução de problemas, a inadequação dos métodos pedagógicos aos estilos de aprendizagem dos alunos, referindo ainda que as linguagens de programação possuem sintaxes adequadas para profissionais mas não para aprendizes inexperientes.

Sloane e Linn (1988) referem que algumas das capacidades exigidas são óbvias, considerando como essenciais a capacidade de resolução de problemas e alguma ideia da matemática subjacente ao processo. Consideram também que um programador deverá ser capaz de usar o computador eficientemente, tem de saber criar um programa num ficheiro, compilá-lo e encontrar os resultados gerados. O programa produzido deverá então ser testado, os bugs encontrados e corrigidos. Porém, segundo estes autores, estas são as competências fáceis de identificar. Consideram também que existem competências menos óbvias, classificadas como “competências de vida”. Os mesmos autores referem ainda que programar não consiste apenas numa única capacidade, não se trata apenas de um conjunto de aptidões mas antes de uma hierarquia de aptidões e um programador necessitará de muitas delas em algum momento da sua vida.

Bereiter e Ng (1991) referem que um aluno que se depare com a aprendizagem de uma hierarquia de competências, geralmente começa pelas de mais baixo nível progredindo gradualmente até às mais exigentes. Os autores concretizam que, no caso da codificação (uma pequena parte de uma competência de programação) implica que os alunos aprendam a sintaxe básica e gradualmente aprendam a semântica, estrutura e finalmente o estilo.

O que sugere que aprender a programar é um processo que exige tempo e maturidade.

Jenkins (2002) argumenta que a programação é normalmente ensinada como assuntos fundamental no início de um curso superior, sendo esse um momento de transição e muitas dificuldades/novidades, pois muitas vezes a transição para a universidade implica pela primeira vez uma vida autónoma, independente e longe dos familiares. Este autor considera que a programação é por si uma matéria difícil quando os alunos estão estáveis, sendo a situação agravada num período de transição.

O mesmo autor refere-se também à falta de motivação devido à imagem negativa apresentada pelos programadores. Existe a imagem pública de um programador como um "desadequado social", o que faz com que seja pouco provável que os alunos aspirem a uma imagem deste tipo. Outros autores referem que a generalidade dos alunos não têm motivação intrínseca para estudar estes assuntos e sem este tipo de motivação dificilmente serão bem sucedidos (Bereiter e Ng, 1991).

Por seu lado, Dunican (2002) afirma que os problemas verificados com os alunos de programação Irlandeses são o produto dos sistemas educativos (primários e secundários) sem módulos de resolução de problemas ou lógica, em qualquer uma das suas disciplinas. Em segundo lugar, estes autores afirmam que outro grande problema se prende com a natureza abstracta da tarefa de programar. Noções como variáveis, tipos de dados, memória dinâmica, entre outros, não têm correspondência na vida do dia-a-dia, e compreender estes conceitos fundamentais de programação não é simples. Em terceiro lugar, as exigências rígidas em termos de sintaxe quando comparada com a natureza inexacta e livre da língua inglesa faz com que muitos alunos não sejam capazes de escrever programas compiláveis bem sucedidos.

3. OS PROBLEMAS DE APRENDIZAGEM DE PROGRAMAÇÃO - A NOSSA PERSPECTIVA

Na nossa perspectiva existe um conjunto de factores que complicam a difícil tarefa de aprender a programar, relacionados com os métodos de ensino, os métodos de estudo e a natureza específica do tipo de matéria.

Relativamente aos métodos de ensino, na prática ainda não existe um verdadeiro ensino centrado no aluno. Os tempos lectivos continuam a ser demasiado "apertados" e as turmas demasiado grandes para promover um ensino personalizado, com um *feedback* e supervisão adequados às necessidades de cada aluno. Adicionalmente, muitos professores continuam a achar que o ensino superior não necessita de pedagogia e que os alunos têm obrigação de se adaptar ao estilo de cada professor. Frequentemente, os professores esquecem-se de diversificar as suas estratégias de forma a contemplar a grande diversidade de pensamentos, compreensões, ritmos e estilos de aprendizagem existentes em cada turma.

A natureza específica da programação, substancialmente diferente do da maioria das disciplinas, implica o ensino de muitos conceitos dinâmicos que é, normalmente, realizado através de materiais de natureza estática (apresentações projectadas, explicações verbais, diagramas, desenhos no quadro, textos, e assim por diante) não promovendo uma plena compreensão da dinâmica envolvida. Também a estrutura curricular das disciplinas de programação está na nossa óptica "virada ao contrário", seguindo uma aproximação *bottom-up* quando pensamos que o ideal seria uma abordagem *top-down*. É prática comum começar por ensinar os detalhes sintácticos de uma linguagem de programação, qualquer que ela seja, antes que os alunos, percebam qual a finalidade e utilidade de aprender programação. Pensamos que a preocupação principal deveria ser antes de mais o desenvolvimento da capacidade de resolução de problemas, aparecendo a linguagem de programação apenas como um veículo para concretizar essa resolução, ou seja, para expressar o algoritmo ou estratégia de resolução.

Relativamente aos métodos de estudo adoptados pelos alunos são também desejáveis diversas mudanças. Os alunos estão frequentemente habituados a disciplinas às quais é possível ser bem sucedido através de abordagens de estudo baseadas em leituras sucessivas, memorização de fórmulas e uma certa mecanização de procedimentos. Porém, a programação impõe um estudo bastante diferente, exigindo prática intensiva, uma verdadeira compreensão dos assuntos e reflexão. Adquirir competências necessárias para programar envolve para além do conhecimento do domínio da programação, um bom *background* de conhecimentos matemáticos e de resolução de problemas. Assistir às aulas e estudar um livro de texto não é o suficiente. Programar exige um intenso trabalho extra aulas. Aos alunos falta

muitas vezes a persistência exigida por problemas cuja solução não se encontra normalmente de forma simples e rápida, como acontece com a maioria dos problemas de programação. É precisamente o conhecimento adquirido na procura de uma resolução difícil, que permite desenvolver estruturas cognitivas valiosíssimas neste domínio.

Porém, pensamos que os problemas de muitos alunos se prendem com um conjunto de dificuldades concretas que levam a que não consigam programar, mesmo quando se esforçam. A primeira dificuldade, frequentemente não percebida pelos alunos, diz respeito à compreensão do problema, pois muitas vezes os alunos "saltam" para a fase de codificação, sem compreenderem completamente o que é pretendido. Tal pode acontecer por dificuldades de interpretação ou devido aos alunos se sentirem demasiado ansiosos para começar a codificar uma solução. É importante que os alunos compreendam completamente os dados do problema e o que é esperado obter como resultado, para que posteriormente possam pensar no algoritmo necessário para essa transformação. No entanto, as grandes dificuldades sentidas pelos alunos são geralmente verificadas na fase seguinte, a construção do algoritmo, o que na nossa opinião é fundamentalmente devido a um défice de capacidade de resolução de problemas manifestado por muitos alunos. Se a resolução de problemas genéricos já implica competências referidas nos últimos patamares da Taxonomia de Bloom (Bloom, 1956), a resolução de problemas de programação exige ainda várias outras competências, algumas das quais têm de estar activas simultaneamente. Associada a esta questão está também um elevado défice de conhecimentos matemáticos básicos manifestado por muitos alunos. Gomes et al. (2006) conduziram algumas experiências a fim de encontrar correlações entre a falta de conhecimentos matemáticos e a carência de competências de programação. Nesta experiência os autores concluíram que os alunos com problemas em aprender a programar apresentavam dificuldades profundas em diversas áreas, tais como cálculo básico e teoria de números ou conceitos geométricos e trigonométricos simples. Os autores relatam também dificuldades em relacionar a descrição textual de um problema com a fórmula matemática que o resolve. Limitações ao nível da abstracção e do raciocínio lógico foram também identificadas. Todos estes problemas levam a que os alunos inexperientes apresentem dificuldades em aplicar conceitos básicos, como estruturas de controlo, para resolver problemas reais, numa fase inicial de aprendizagem de programação.

Outro aspecto importantíssimo, corresponde à última etapa da programação, muitas vezes negligenciada pelos alunos, o teste e reflexão sobre o problema e sua solução. Geralmente os alunos, não fazem o teste/simulação da solução construída ou quando muito fazem-no muito superficialmente ou apenas para um conjunto reduzido de testes sem verificação de casos limite. A reflexão acerca da forma de resolução de determinado problema, bem como o questionamento sobre novas propostas de solução em função de alterações no enunciado, constituiria uma mais-valia para a aprendizagem dos alunos se fosse uma prática habitual.

Por último, também consideramos, tal como outros autores já referidos, a falta de motivação apresentada por muitos alunos como uma preocupação a não descurar.

4. PROPOSTA

Como resolver ou minimizar cada um dos problemas referidos? Pensamos que o desenvolvimento e utilização de um ambiente computacional com características inovadoras poderá resolver parte dos problemas focados.

Ao longo dos tempos foram desenvolvidos diversos tipos de sistemas computacionais de apoio à aprendizagem da programação, recorrendo a representações visuais/animações de algoritmos, linguagens de programação baseadas em ícones, Sistemas de Tutores Inteligentes, micromundos de aprendizagem, entre outros.

O nosso grupo também já desenvolveu algumas ferramentas destinadas a apoiar a aprendizagem básica de programação nomeadamente o VIP (Mendes e Mendes, 1988) o SICAS (Gomes e Mendes, 2001), o PROGUIDE (Areias, 2007), o OOP-ANIM (Esteves e Mendes, 2004) e o SICAS-COL (Rebelo, 2007). Mas com um conjunto tão vasto de alternativas, que segundo os seus autores representam um contributo valioso para o ensino/aprendizagem de programação, porque razão é que os problemas subsistem? Porque é que essas ferramentas não são amplamente utilizadas

com resultados satisfatórios? Na realidade, e pelos estudos já efectuados, parece que nenhuma das ferramentas disponíveis supre completamente as exigências da aprendizagem de programação. Como pensamos que o problema principal reside na incapacidade de os alunos resolverem problemas, ou seja, construírem algoritmos que os resolvam, criou-se uma aplicação, SICAS (Sistema Interactivo para Construção de Algoritmos e sua Simulação) (Gomes, 2002), cuja preocupação principal era fornecer um ambiente onde os alunos não apenas compreendessem as diversas fases de um algoritmo já concebido, mas sobretudo que permitisse que o aluno concebesse, testasse, experimentasse, alterasse e corrigisse os seus próprios algoritmos. Contudo, as avaliações efectuadas com o SICAS demonstraram que a abordagem utilizada não é suficiente para contemplar todos os alunos. Por um lado, é um ambiente que não promove de igual forma todos os estilos de aprendizagem, mas antes favorece os alunos marcadamente visuais em detrimento dos verbais. Por outro lado, constitui uma abordagem útil apenas para aqueles alunos que quando confrontados com o enunciado de um problema conseguem iniciar a sua resolução e construir uma primeira solução, mesmo que não completamente correcta. Mas, na realidade, existem muitos alunos com enormes dificuldades que perante o enunciado de um problema simples nem sequer conseguem chegar a uma primeira proposta de solução. Nestes casos o SICAS é de pouca utilidade. Assim, a nossa proposta centra-se em primeiro lugar num ensino personalizado, que adapte as actividades a cada aluno de acordo com o seu estado cognitivo, ritmo e estilo de aprendizagem. Um aspecto central do sistema em desenvolvimento consiste em diagnosticar o estilo de aprendizagem preferencial de cada aluno, de modo a condicionar a forma de apresentação dos problemas/actividades em todo o ambiente. Existem diferentes modelos para determinar o estilo de aprendizagem de um indivíduo, nomeadamente "The Myers-Briggs Type Indicator (MBTI)" (Myers e McCaulley, 1985), "The Kolb's Learning Style Model" (Kolb, 1985), "The Felder-Silverman Learning Style Model" (Felder, 1988), entre outros. Porém, temos vindo a fazer diversas experiências com o "The Felder-Silverman Learning Style Model" adoptando o tipo de inquérito nele proposto para diagnosticar os estilos de aprendizagem dos alunos. A principal razão da escolha deste modelo prende-se com o facto de ter sido desenvolvido a pensar em alunos de engenharia, também o nosso público-alvo, para além de possuir um inquérito *on-line* que facilmente permite caracterizar um indivíduo. No entanto, outros modelos não estão ainda excluídos.

Pretende-se então construir um sistema centrado no desenvolvimento da capacidade de resolução de problemas, sendo baseado numa aproximação construtivista da aprendizagem, onde o aluno aprende fazendo, experimentando e deduzindo, construindo progressivamente o seu próprio conhecimento. Uma parte fundamental do ambiente, consiste na incorporação de vários tipos de actividades lúdicas e jogos lógicos que, de uma forma atractiva e estimulante, permitam desenvolver a capacidade de resolução de problemas nos alunos. A metodologia proposta é constituída por três fases. A primeira trabalha a resolução de problemas de diversos domínios (quebra-cabeças simbólicos, quebra-cabeças lógicos, jogos e charadas, problemas simples de aritmética e geometria, entre outros) não tratando directamente de algoritmos ou de programação. Em seguida, e gradualmente, o sistema mostrará ao aluno a utilidade da programação, com aplicação dos conhecimentos adquiridos na fase anterior. Finalmente, o objectivo passa pela construção de algoritmos, pretendendo transformar a formalização desenvolvida em procedimentos sistemáticos. Gradualmente, os problemas apresentados ao aluno passarão a exigir soluções mais elaboradas, nas quais cada vez mais estará inerente o acto de explicitar procedimentos. Cada uma das fases apontadas é sempre aplicada de acordo com o estado actual de conhecimento do aluno e do seu estilo preferencial de aprendizagem.

4.1. 1ª Fase – Resolução de Problemas

Na primeira fase, a grande questão prende-se com o tipo de problemas a apresentar aos alunos. Será que qualquer tipo de problema desenvolve as capacidades pretendidas? Que capacidades, competências ou funções cognitivas se pretendem realmente exercitar? De forma a responder a estas questões pesquisou-se a literatura sobre as competências necessárias para resolver problemas de programação. De acordo com diversos autores estas envolvem, habilidades matemáticas, raciocínio analógico, raciocínio condicional; pensamento procedimental e raciocínio temporal (Pea e Kurland, 1984) ou raciocínio analítico, raciocínio quantitativo, raciocínio analógico, raciocínio combinatório (OCDE, 2003), entre outros. A psicologia

oferece diversos instrumentos de avaliação para detectar quais as funções cognitivas ou tipos de competências/raciocínios que um indivíduo tem em défice. Adequados à faixa etária e competências necessárias dos alunos em causa destacam-se as Matrizes Progressivas de Raven (Simões, 1995) e o PARC (Ribeiro e Almeida, 1999). O PARC, surgiu como complemento à BPRD (Bateria de Provas de Raciocínio Diferencial) (Almeida, 1995) mas para jovens e jovens adultos incluindo, para além de provas de raciocínio, outros processos cognitivos como a compreensão e o pensamento divergente. No entanto, estamos a iniciar um conjunto de experiências com o ABI – Aptidões Básicas para Informática (Cruz e Fonseca, 2002). Esta bateria de exercícios foi concebida especificamente para a avaliação de candidatos ou de profissionais da área de informática. Avalia a compreensão verbal (prova de compreensão verbal do tipo "sinónimos"), a compreensão de problemas e de conceitos matemáticos (avalia a capacidade do sujeito para manipular símbolos matemáticos e para resolver problemas numéricos), a atenção e resistência à monotonia (avalia a atenção concentrada, mediante uma tarefa de detecção de erros num determinado contexto), o raciocínio lógico (prova de raciocínio em que os sujeitos devem encontrar, nas soluções possíveis, o número que continuaria a série apresentada), a capacidade de classificação e de análise (avalia um aspecto específico da atenção: a capacidade para localizar elementos que estão misturados com outros e assinalar o código correspondente) e a capacidade de organização de fases lógicas, também denominada de prova de diagramas (avalia a capacidade para analisar um problema e para organizar soluções numa série de etapas lógicas).

Adicionalmente também se pretende aplicar Testes de auto-estima e motivação. A sua pertinência justifica-se dado que os alunos com dificuldades escolares tendem a ter uma imagem pessoal menos positiva, pouco favorável à aprendizagem e desempenho. Logo, ao identificarmos défices desta ordem podemos realizar treinos cognitivos que visam proporcionar mudanças no âmbito das expectativas e das percepções pessoais de competência (Almeida e Balão, 1996 cit in Cruz e Fonseca, 2002), contribuindo desta forma para potenciar a aprendizagem. A motivação é um factor que não pode, de forma alguma, ser esquecido. É fundamental perceber os sentimentos e imagens dos alunos em relação às suas capacidades, à sua realização cognitiva e à sua aprendizagem. Isto porque uma auto-estima, auto-conceito e expectativas de eficácia pouco positivas vão ter influência na motivação, entusiasmo e persistência do aluno na realização das tarefas implementadas no programa.

Pretendemos ainda aplicar inventários de atitudes e comportamentos habituais de estudo, como por exemplo o IACHE. Um aspecto relevante para o sucesso na aprendizagem de programação prende-se com os métodos de estudo. Desta forma, pretende-se aplicar um instrumento deste tipo para perceber o tipo de abordagem da aprendizagem seguido prioritariamente pelos alunos com dificuldades: profunda (motivação intrínseca), de alto-rendimento (competição e maximização do sucesso) ou superficial (motivação extrínseca, aprendizagem baseada na simples memorização dos conteúdos).

A aplicação de um dado teste cognitivo, permite identificar determinadas lacunas cognitivas no aluno, após o que se pretende propor estratégias/testes para treinar as funções cognitivas em falta. A abordagem do treino cognitivo é uma abordagem do processamento da informação com a finalidade de compreender as capacidades mentais, de um ponto de vista de treinabilidade dos processos cognitivos. Existem programas que treinam as funções cognitivas básicas e superiores e outros que delimitam o campo de actuação a processos cognitivos mais específicos. No entanto, alguns autores defendem que os resultados deste tipo de abordagem só podem ser duráveis se houver um treino ao nível das metacomponentes (funções executivas, de ordem superior) e ao nível das componentes de desempenho (processos de ordem inferior) (Cruz e Fonseca, 2002). A análise e aplicação dos diversos testes referidos tem como intuito aferir o tipo de problemas a incluir no sistema. A incapacidade demonstrada pelo aluno em resolver um problema de determinado tipo, sugere a inclusão no sistema de actividades de treino cognitivo que também se encontram em elaboração.

4.2. 2ª Fase – Demonstração da programação através de um jogo

Na segunda fase, pretende-se mostrar aos alunos para que serve e como se faz um programa. Para tal, será ilustrado de forma interactiva o desenvolvimento e implementação de um jogo, procurando utilizar-se o carácter lúdico para gerar um nível de motivação superior. O jogo escolhido será o jogo do galo, por ser um jogo bem conhecido e por ter sido o 1º jogo computacional. A estratégia de compreensão usada é *Top-Down*, começando

pelos conceitos mais gerais de um programa e através de um processo de análise - por decomposição ou refinamentos sucessivos chegar-se ao detalhe. Começa-se assim pelo conhecimento do problema, um jogo conhecido do aluno. O jogo será ilustrado recorrendo a 3 robots (bonecos animados) com funções distintas, nomeadamente:

- "robot-A"- que irá ilustrar o jogo do ponto de vista do jogador.
- "robot-B" – que irá ilustrar a forma lógica (programação) de representação do jogo.
- "robot-C" – que irá ilustrar a forma física (armazenamento interno) de representação do jogo.

A estratégia geral consiste em introduzir o robot-A que dialogará com o aluno sobre os diversos elementos constituintes do jogo, as diversas regras a respeitar, de forma a assegurar uma completa compreensão e correcto planeamento do jogo. Durante este processo o robot-A utilizará várias metáforas, recorrendo a animações que mostram exemplos reais conhecidos do aluno que lhe facilitem a compreensão de diversos conceitos úteis e permitam mais facilmente fazer a transição para o campo da implementação (função do robot-B). É então da responsabilidade do robot-B a explicação de como fazer certas implementações a um nível lógico, nomeadamente quais as estruturas de programação que permitem representar o tabuleiro, os jogadores, situação de final de jogo, entre outras. Numa fase final e para um melhor entendimento das questões computacionais, será possível, através do robot-C mostrar a forma como os diversos acontecimentos se processam internamente (ao nível da máquina).

Desta maneira, o aluno pode concentrar-se inicialmente na lógica geral do programa, sem se preocupar com os pormenores sintácticos das instruções individuais. Este processo de planeamento pode então ser repetido várias vezes, com pormenores de programação adicionados em cada fase.

Cada uma das fases inclui um questionamento permanente, onde são aplicadas as etapas mencionadas na literatura para uma correcta resolução de problemas, a dificuldade gradual de apresentação de situações de acordo com a Taxonomia de Bloom, não esquecendo os Estilos de Aprendizagem preferenciais de cada aluno.

4.3. 3ª Fase – Treino de programação

A implementação desta fase pode ser conseguida, por exemplo utilizando o SICAS ou um ambiente similar. O SICAS possibilita, essencialmente, dois tipos de cenários: edição/resolução de problemas e execução/simulação de resoluções previamente construídas pelo aluno. No primeiro cenário, o aluno pode construir algoritmos através de representações visuais – fluxogramas – recorrendo a simbologia gráfica que representa as principais estruturas necessárias à construção de um algoritmo. No segundo cenário, o utilizador pode simular a execução das resoluções construídas, analisando-as com o detalhe e ritmo desejado.

O SICAS apresenta um conjunto de possibilidades de utilização educativa que nos parecem relevantes. Destacamos a possibilidade de os alunos construírem e simularem os seus próprios algoritmos, analisando os respectivos resultados e corrigindo aspectos eventualmente menos conseguidos. Esta é uma actividade de grande importância para a aprendizagem dos fundamentos da programação, objectivo primeiro do desenvolvimento deste ambiente. Apesar de ter sido concebido para uma utilização independente, este ambiente pode também suportar actividades em contexto de sala de aula, mesmo que o professor pretenda levar a cabo um conjunto de actividades mais controladas e específicas. O SICAS pode evidentemente ser utilizado em qualquer outro local, fora do horário curricular, no âmbito do estudo autónomo fundamental na aprendizagem da programação.

Outro aspecto importante da utilização do SICAS reside na possibilidade de, em utilização autónoma e sem preocupações classificativas, o aluno auto-avaliar os seus conhecimentos através da simulação e teste das suas resoluções. Em particular, a possibilidade de verificar que o seu algoritmo se comporta correctamente com os testes especificados pelo professor (dados de entrada e resultados esperados) pode apresentar uma credibilidade superior, conferindo ao aluno um grau de confiança mais elevado no sistema e nas suas próprias capacidades.

Outro aspecto importante reside na possibilidade de permitir ao professor criar conjuntos de exercícios resolvidos, constituindo assim mais um auxiliar para o estudo dos seus alunos. Claro que podem ser utilizadas diversas abordagens pedagógicas, como seja fornecer resoluções correctas,

incorrectas ou ainda incompletas. Apesar de acharmos que analisar problemas resolvidos não se traduz inevitavelmente num aumento da capacidade de solucionar novos problemas, pensamos que pode ser interessante permitir aos alunos fazer alterações às soluções existentes ou propor e testar soluções alternativas. Também a possibilidade de dar aos alunos algoritmos errados, em especial com o tipo de erros lógicos que o aluno em causa habitualmente apresenta, e pedir-lhe que procure e corrija esses erros, pode apresentar um alto valor educativo nesta área.

Outra característica importante do SICAS é a possibilidade de os alunos compararem algoritmos diferentes para um mesmo problema e verificarem quando é que uns apresentam um desempenho superior aos outros, interiorizando assim gradualmente técnicas eficazes de programação. O ambiente pode também ser utilizado numa inversão de papéis, podendo ser pedido ao aluno para indicar o enunciado de um problema cuja resolução lhe seja facultada (e que ele pode analisar com o SICAS).

Destacamos ainda a possibilidade de o professor colocar problemas aos alunos e verificar o seu grau de proficiência através da análise das soluções por eles propostas. De acordo com esta perspectiva, é possível afirmar que o SICAS permite avaliar e individualizar as actividades desenvolvidas pelos alunos. Com este conhecimento, o professor pode propor actividades de acordo com os níveis actuais de conhecimentos de cada aluno, evitando propor problemas demasiado fáceis ou difíceis, o que geralmente se traduz em desmotivação dos alunos. Este aspecto acrescenta um conjunto de valores importantíssimo aos métodos de ensino tradicionais, na medida em que possibilita uma actividade de ensino/aprendizagem mais personalizada, contribuindo para que os alunos possam aprender ao seu próprio ritmo, aumentando a sua motivação, pois muitas vezes os factores que mais contribuem para o desinteresse dos alunos nas salas de aula é a sua total incapacidade de acompanhamento dos exercícios que estão a ser abordados. É claro que este objectivo poderia ser atingido por outros meios, mas pensamos que o SICAS pode proporcionar algum suporte a esta abordagem, necessariamente mais trabalhosa para o professor.

Prevê-se, no entanto, o melhoramento e inclusão de novas funcionalidades, de que se destaca, por exemplo, proporcionar o desenvolvimento de actividades algorítmicas que contemplem não apenas os alunos visuais (através de fluxogramas) mas também os verbais (através de pseudocodigo).

5. CONCLUSÃO

Face às dificuldades apresentadas pelos alunos inexperientes em programação, expomos neste artigo uma proposta que visa contribuir para ultrapassar parte destas dificuldades, apresentando uma abordagem que:

- Contribua para o aumento da motivação do aluno, através de um ambiente lúdico e estimulante, com companheiros/tutores sempre bem dispostos e muito pacientes.
- Inclua animações e modelos dinâmicos que melhor representem os vários conceitos de programação.
- Apresente e proponha actividades de acordo com o estilo de aprendizagem preferencial de cada aluno e de acordo com o seu ritmo e estado cognitivo.
- Ofereça uma abordagem gradual de apresentação da programação treinando, em primeiro lugar competências básicas de resolução de problemas, de planeamento de soluções e mostrando a utilidade da programação, deixando para mais tarde os detalhes sintácticos das linguagens de programação
- Minimize, pelo menos inicialmente, os complexos detalhes sintácticos de uma linguagem de programação.
- Utilize metáforas e exemplos concretos conhecidos do aluno para ajudar a diminuir a carga abstracta inerentemente associada à programação.
- Exiba um ambiente estruturado que assegure metodologias de estudo correctas que levem à reflexão e questionamento permanente.
- Permita um treino intensivo de resolução de problemas, “obrigando” o aluno a seguir todas as etapas para uma correcta resolução de problemas, que se resumem à compreensão do problema, caracterização do problema, representação do problema, solução do problema e reflexão sobre a solução obtida.

- Permita um treino intensivo de conhecimentos matemáticos e lógicos úteis à programação.
- Possibilite que os alunos ganhem uma certa experiência em programação, dando-lhe sugestões para atingir determinadas soluções, propondo actividades diversificadas (programas completos para os alunos analisarem, programas que contenham erros lógicos habitualmente cometidos pelos alunos, programas incompletos para completar, entre outras actividades), apresentando modelos de forma a que os alunos adquiram as melhores práticas de programação.

6. REFERÊNCIAS BIBLIOGRÁFICAS

ALMEIDA, L. (1995). Bateria de Provas de Raciocínio Diferencial (BPRD). In L. Almeida, M. Simões & M. Gonçalves (ed.), *Provas psicológicas em Portugal*, vol.1, p.19-28. Braga: APPORT.

ALMEIDA, E. S., COSTA, E. B., BRAGA, J. D. H., SILVA, K. S., PAES, R. B. e ALMEIDA, A. A. M. (2002). AMBAP: Um Ambiente de Apoio ao Aprendizado de Programação. I*n X Workshop sobre Educação em Computação, Florianópolis. Anais do WEI 2002/ SBC2002.*

AREIAS, C. (2007). *ProGuide: Sistema de acompanhamento na resolução de problemas básicos de programação*. Tese de Mestrado em Engenharia Informática, Universidade de Coimbra.

BEREITER, C. and NG., E. (1991). Three Levels of Goal Orientation in Learning. *In Journal of the Learning Sciences*, nº 3, (vol. 1), 243-271.

BLOOM, B. S. and KRATHWOHL, D. R. (1956). *Taxonomy of Educational Objectives, Handbook I: Cognitive Domain.* Longmans, Green and Company.

BRIGGS-MYERS, I. and MCCAULLEY, M. H. (1985). *Manual: A Guide to the Development and Use of the Myers-Briggs Type Indicator*. Palo Alto, CA: Consulting Psychologists Press.

BYRNE, P. and LYONS, G. (2001). The Effect of Student Attributes on Success in Programming. In *Proceedings of the 6th Annual Conference on Innovation and Technology in Computer Science Education - ITiCSE 2001, United Kingdom, p.49-52*.

CRUZ, V. and FONSECA, C. (2002). *Educação Cognitiva e Aprendizagem*. Porto Editora, Porto.

CRUZ, M. Aptidões Básicas para Informática (ABI), Available in http://www.cegoc.pt/testes/catalogo/testes.asp?Tema=25

DIJKSTRA, Edsger W. (1989). On the Cruelty of Really Teaching Computing Science. *In Communications of ACM*, Issue 12, (vol.32), 1398-1404.

DUNICAN, E. (2002). "Making The Analogy: Alternative Delivery Techniques for First Year Programming Courses". In *14th Workshop of the Psychology of Programming Interest Group, Brunel University, p.89-99, In J. Kuljis, L. Baldwin & R. Scoble (Eds).*

ESTEVES, M. and MENDES, A. (2004). A Simulation Tool to Help Learning of Object Oriented Programming Basics. In *Proceedings of the 34th ASEE/IEEE Frontiers in Education Conference*.

FELDER, R. M. (1988). Learning and Teaching Styles in Engineering Education. *In Journal of Engineering Education*, nº 7, (vol. 78), 674-681.

GOMES, A. e MENDES, A. J. (2001). SICAS: Interactive system for algorithm development and simulation. In Manuel Ortega y José Bravo (Ed.), Computers and Education in an Interconnected Society, Kluwer Academic Publishers, p.159-166.

GOMES, A. (2002). *Ambiente de suporte à aprendizagem de conceitos básicos de programação*. Tese de Mestrado em Engenharia Informática, Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

GOMES, A., CARMO, L., BIGOTTE, E. and MENDES, A. J. (2006). "Mathematics and programming problem solving", *in Proceedings of 3rd E-Learning Conference – Computer Science Education, Coimbra, Portuga*l *[CD-ROM]*.

JENKINS, T. (2002). On the difficulty of learning to program. In *Proceedings of 3rd Annual LTSN_ICS Conference (Loughborough University, United Kingdom, August 27-29, 2002). The Higher Education Academy, p.53-58.*

JÚNIOR, J. C. R. P. e RAPKIEWICZ, C. E. (2004). "O Processo de Ensino e Aprendizagem de Algoritmos e Programação: Uma Visão Crítica da Literatura". In *III Workshop de Educação em Computação e Informática do estado de Minas Gerais (WEIMIG' 2004). Belo Horizonte, MG, Brasil.*

JÚNIOR, J. C. R. P., RAPKIEWICZ, C. E., Delgado, C. e Xexeo, J. A. M. (2005). "Ensino de Algoritmos e Programação: Uma Experiência no Nível Médio". In *XIII Workshop de Educação em Computação (WEI'2005). São Leopoldo, RS, Brasil*.

KOLB, D. A. (1985). *Learning Style Inventory: Technical Manual*. McBer and Company, Boston.

MENDES, A. J. e MENDES, T. (1988). VIP – A Tool to VIsualize programming examples. *In Proceedings of Education and Application of Computer Technology*, p.131-140.

Motil, J. and Epstein, D. (2000). JJ: a Language Designed for Beginners (Less Is More). Available at http://www.ecs.csun.edu/jmotil/TeachingWithJJ.pdf

Myers, I. B. and McCaulley, M. H. (1985). *Manual: A Guide to the Development and Use of the Myers – Briggs Type Indicator*. Palo Alto, CA, Consulting Psychologists Press.

OECD (Organisation for Economic Co-operation and Development). Learning for tomorrow's world. First results from PISA 2003, Paris, available in http://www.pisa.oecd.org/dataoecd/38/30/33707234.pdf

PEA, R. D. and KURLAND, D. M. (1984). On the Cognitive and Educational Benefits of Teaching Children Programming: A Critical Look. *New Ideas in Psychology*, nº 2, p.147-168.

PERKINS, D. N., SCHWARTZ, S. and SIMMONS, R.; (1988). Instructional Strategies for the Problems of Novice Programmers. In R. E. Mayer (ed.), *Teaching and Learning Computer Programming*, p.153-178. Hillsdale, NJ: Lawrence Erlbaum Associates.

REBELO, B. (2007). *SICAS-COL - Um Sistema Colaborativo para Aprendizagem Inicial da Programação*. Tese de Mestrado em Engenharia Informática, Universidade de Coimbra.

RIBEIRO, I. & ALMEIDA, L. (1999). Provas de avaliação da realização cognitiva (P. A. R. C.*)*. Iin M. Simões, M. Gonçalves & L. Almeida (ed.), *Testes e provas psicológicas em Portugal*, vol.2, p.71-79. Braga: APPORT/SHO.

Simões, M. (1995). Portuguese standardisation of the Raven's Coloured Progressive Matrices: An overview of item analyses, reliability, validity and norming studies. In *Proceedings of 3rd European Conference on Psychological Assessment, Trier, Germany.*

SLOANE, K. D. and LINN, M. C. (1988). Instructional Conditions in Pascal Programming Classes. In R. E. Mayer (ed.), *Teaching and Learning Computer Programming*, p.207-235. Hillsdale, NJ: Lawrence Erlbaum Associates.

Abstract: The high failure rate in introductory programming courses, where basic programming concepts are taught, is a universal problem that has motivated several authors to investigate the causes of those difficulties. Several tools have been proposed to help students learn programming. Although some of these tools have been reported to have a positive effect in student learning, the problem remains almost the same.
From our point of view there are several reasons that cause that problem. We consider that the study and teaching methods are not suitable, the students lack previous preparation on problem solving. We also consider that the nature of the subject is complex and that students have lack of motivation in studying for these subjects.
We think that the traditional methodologies to teach and learn these subjects are not enough nor appropriate. So we propose a new tool, focused on problem solving, providing a set of suitable activities according to the students' cognitive needs and knowledge level, also supporting students with different backgrounds and learning styles.

Key-words: Programming Learning, Learning Styles, Psychology of Programming.

Texto

- Submetido em Fevereiro de 2008
- Aprovado em Março de 2008

Como citar este texto:

GOMES, A., HENRIQUES, J., MENDES, A. J. (2008). "Uma proposta para ajudar alunos com dificuldades na aprendizagem inicial de programação de computadores". "In *Educação, Formação & Tecnoloigias;* vol.1(1), pp. 93-103. Disponível em http://eft.educom.pt